ALLAN THRAEN | 3 years ago | PDF |

Tips and Tricks CMS

WYAM - DYNAMICALLY CREATING A STATIC SITE

Wyam is a really cool .NET Core based tool to generate static websites. I took it out for a test-run, using this blogs RSS feed as a content source.

Dynamically updated, static sites

Static sites are getting a lot of buzz these days. At first you might think - "Stop! Wait! 1996 called, and want's it internet technology back". But don't worry. We're not going to boot up an old floppy with Microsoft Frontpage along with your favorite FTP client and start using the <blink> tag all over ;-) Todays static websites are typically powered by content management systems, and pretty much the same set of tools you are used to. React, Node, Rails, Python, .Net Core, Razor, Yaml, Markdown - the list goes on.

What it basically boils down to, is the fact that many (if not most) websites today could easily be static - since they are primarily one-way communication, and whatever dynamic aspects are there (personalization, recommendations, forms, search) are typically powered by client script in the browser and/or 3rd party hosted services. In fact, I would argue that most sites, where you have no problem turning on full-output cache, could probably be served statically - and many are, often through a CDN layer.

This new world of static site is being pioneered by technologies such as Gatsby, Netlify, Contentful, Wyam and others. And it's clear to see why. The advantages of static sites are many:

- Performance. It's hard to rival the speed of just serving up flat files.
- Cost. Webhotels for flat files are easy to find and they are incredibly cheap. You can even host your site on a pure Azure Blob account with a custom domain attached. No expensive database-server.
- Resilience. No database to get corrupt, no assembly hell that can break on upgrade, no multi-server setup that needs to be in sync. And disaster failover is as easy as 'xcopy'.
- Security. Less complicated setup, means fewer places with potential security breaches. And should the repo for the flat files get compromised, you just redeploy.

Some time ago, I blogged about an example where I used Episerver CMS to automatically create a static failover site on Azure Blob storage. Since Episerver is both a platform for building websites on, as well as a content management system, that worked out quite well. But with many other static site builders, the flow is quite different...

Introducing Wyam

Wyam is a really cool static content toolkit, built on .NET Core. It's really easy to use, once you wrap your head around how it works (which is quite a bit different from what cms geeks like me are used to).

Basically, the key concept is that Wyam takes a set of **input files** (which doesn't have to be physical files as you can have virtual file providers), loads them into **documents** (which consist of both content and metadata) and then let's you describe one or more **pipelines**, which consists of a series of **modules** that enrich/split/merge the documents - eventually leaving your with the ability to write a set of **output files**. It sounds more complicated than it is. Think of it as a content conversion engine. You typically put content + metadata + renderings into one end - and out the other comes html ready to be served. Although it's of course not limited to html. In theory any kind of content could be moulded to any other kind.

The CLI is really easy to install, just type:

*dotnet tool **install** -g Wyam.Tool*

and you'll get it. You can scaffold files to get, for example a blog, following a 'recipe' like typing *wyam new --recipe Blog*. In a folder, and it will create a set of input files for a blog, and a config file to build them.

But the key is in the *config.wyam* file. It's essentially a csharp based script file, where you describe which nuget packages you need, and how you want to compose your pipelines to get the right output.

Building a blog from a blog

I figured a fun way to test this out would be to see how hard it would be to setup a config file that dynamically would fetch content from my blog (through my rss feed), render each post with razor, and an overview page with links to each post.

Turns out, it requires 2 pipelines. First a pipelines for the posts, and secondly a pipeline for the overview page.

```
Pipelines.Add("Posts",
  Download("https://www.codeart.dk/blog/rss/"),
   Xml("//item"),
  Merge(ReadFiles("post/post.cshtml")),
  Razor(),
  Index(),
  Title(@doc["title"]),
   Meta("PostFile",string.Format(@"posts\{0}.html", @doc["Index"])),
  WriteFiles((string)@doc["PostFile"])
);

Pipelines.Add("Home",
```

```
    ReadFiles("home.cshtml"),
    Razor(),
    WriteFiles(".html")
);
```

As you can see, it's pretty straight forward:

1. First I download the rss feed.
2. Then I use the XML module to split the document with the xml feed into many documents, based on the xpath selector //item
3. Now, it's time to merge those files together with the razor file I created to show the posts.
4. And execute the razor rendering engine
5. Now, I assign each document an index number as metadata
6. Set their title
7. I add a metadata field called "PostFile" with the file number I want the files to have. In this case I'm using their index number + ".html"
8. And finally I write the files

As I already have all the documents loaded in memory, all I have to do for the overview page is to load the razor code and render it.

The config file should be in the root of the working folder, the razor files (and others) in a subfolder called "input".

To built it, I just run *wyam* in the working folder. Another trick is to run *wyam -p -w*, where the 'p' argument will start a preview webserver on port 5080 and 'w' will watch the input folders and rebuild if I change any of the files.

Now, I'll admit that my razor in this case isn't all the pretty - but that's not the point :-) Full Gist is below.

In future posts I plan to explore several other interesting aspects around wyam, like:

- How do I use it with content from a CMS
- How do I setup a cloud-based build-flow with Wyam, so my code can live in Github, my content can live in Episerver or Contentful and my site running on Azure Blob storage can be constantly updated (without cheating and using netlify :-))
- What other fun stuff can Wyam convert.

```
<html>
<head>
<title>@ViewBag.Title</title>
</head>
<body>
@RenderBody()
</body>
</html>
```

_Layout.cshtml hosted with ♥ by GitHub view raw

```
@{
    Layout = "_Layout.cshtml";
}
```

_ViewStart.cshtml hosted with ♥ by GitHub view raw

```
// This directive installs packages from NuGet and is what you would normally use
#n Wyam.Razor

// This directive loads modules directly from the local build and is used for testing
// NOTE: If running the example against modules built from source, call wyam.exe from the Wyam.Examples.Tests/bin/Debug folder
#a **/*

// Normalize the culture for the examples so they produce the same output regardless of system culture
System.Globalization.CultureInfo.DefaultThreadCurrentCulture
    = System.Globalization.CultureInfo.CreateSpecificCulture("en-US");

Pipelines.Add("Posts",
 Download("https://www.codeart.dk/blog/rss/"),
 Xml("//item"),
 Merge(ReadFiles("post/post.cshtml")),
  Razor(),
  Index(),
 Title(@doc["title"]),
 Meta("PostFile",string.Format(@"posts\{0}.html", @doc["Index"])),
 WriteFiles((string)@doc["PostFile"])
 );

Pipelines.Add("Home",
 ReadFiles("home.cshtml"),
  Razor(),
 WriteFiles(".html")
 );
```

config.wyam hosted with ♥ by GitHub view raw

```
@{ViewBag.Title="Blog posts";}

<h1>Blog posts from CodeArt</h1>
@foreach(var doc in Documents["Posts"].OrderBy(x => x["Title"]))
{
    <ul>
        <li><a href="@doc["PostFile"]">@doc["Title"]</a> - @doc["pubDate"]</li>
    </ul>
}
```

home.cshtml hosted with ♥ by GitHub view raw

```

@{
    ViewBag.Title=Metadata["Title"];
}

<h1 style="color:red">@Metadata["Title"]</h1>
<h2>@Metadata["pubDate"]</h2>
<hr />
<p>@Metadata["description"]</p>

<a href="@Metadata["link"]">Read more...</a>
```

post.cshtml hosted with ♥ by GitHub view raw

Tips and Tricks CMS

RECENT POSTS

CodeArt ApS
Teknikerbyen 5, 2830 Virum, Denmark
Email: info@codeart.dk
Phone: +45 26 13 66 96
CVR: 39680688